Remarks

I. The Impropriety of the Final Rejection

On August 19, 2008, in response to the Board's complete Reversal of June 16, 2008, the Examiner reopened prosecution with a new Final Rejection over the same reference that he used in the previous Final Rejection that was Reversed by the Board.

The Examiner stated, in making the rejection final, "Applicant's previous amendment, which was filed 11/17/05, necessitated the new ground(s) of rejection presented in this Office action. Accordingly, THIS ACTION IS MADE FINAL."

Applicants respectfully object to the finality of the rejection. MPEP §706.07(a) states, in pertinent part:

> Under present practice, second or any subsequent actions on the merits shall be final, except where the examiner introduces a new ground of rejection that is neither necessitated by applicant's amendment of the claims, nor based on information submitted in an information disclosure statement filed during the period set forth in 37 CFR 1.97(c) with the fee set forth in 37 CFR 1.17(p).

Applicants respectfully disagree that "Applicant's previous amendment, which was filed 11/17/05, necessitated the new ground(s) of rejection presented in this Office action." This Final Rejection was not necessitated by applicant's amendment of almost three years ago, but by the Examiner's refusal to accept the Decision on Appeal of about two months before the Final Rejection, which completely Reversed the Examiner's previous Final Rejection. Had this "new ground of rejection" been "necessitated by applicant's amendment," it would have been made a few years ago.

Moreover, applicants' note that MPEP §706.07(a) cautions the Examiner against the tactics he is employing, stating in pertinent part:

> The examiner should never lose sight of the fact that in every case the applicant is entitled to a full and fair hearing, and that a clear issue between applicant and examiner should be developed, if possible, before appeal.

By making this latest rejection Final without giving applicants a full and fair hearing before they are faced with the time and cost of another Appeal, the Examiner is attempting to disguise the embarrassment of the complete Reversal with tactics designed to limit applicants' response. This is not an appropriate procedure for an Examiner that is supposed to be impartial, and instead simply highlights the lengths to which this Examiner will go to reject the pending claims.

In addition to the impropriety of making this rejection final, applicants respectfully assert that the Examiner has flouted various Patent Office Rules by reopening prosecution after the complete Reversal and over the same reference, without approval at least from the Technology Center (TC) Director, as noted in the Petition mailed September 30, 2008.

II. 35 U.S.C. §103

Claims 1-8, 10-14, 16-27, 29-40 and 42 stand Finally Rejected under 35 U.S.C. §103(a) as allegedly being unpatentable over U.S. Patent No. 6,345,302 to Bennett et al. ("Bennett"). Regarding claim 1, the Final Rejection states:

> As to claim 1, Bennett teaches the invention as claimed including: a system for communication by a local host that is connectable by a network to a remote host [e.g., 276, 1000, Fig.10], the system comprising:
>
> a communication processing device (CPD) [2000, Fig. 3, which is a network interface card] that is integrated into the local host to connect the network and the local host, said CPD including hardware configured to analyze Internet Protocol (IP) and Transmission Control Protocol (TCP) headers of network packets [Abstract; col.15, lines 52-61; i.e., verify the TCP checksum, which is entered in field of header (see 322, Fig. 7)]; and
>
> a central processing unit (CPU) [10, Fig.3] running protocol processing instructions in the local host to create a TCP connection between the local host and the remote host [col.4, lines 23-50; note that the "TCP connection" is being construed as a physical connection established for transmission of a TCP packet. Since TCP process 91 of Fig. 2B is designed for constructing a TCP packet, the execution of process 91 is in a way "running protocol processing instructions"], said CPU providing to said CPD an IP address and a TCP port that correspond to said connection [e.g., col.4, lines 7-22; col.16 lines 19-35; Figs. 2A, 6-7; note that each encapsulated TCP packet (which contains IP and TCP port information) that is sent out from the CPU also flows through the network card],
>
> wherein said CPD and said CPU are configured such that a message transferred between the network and the local host is generally processed by said CPD instead of said CPU when said CPD controls said connection and said message corresponds to said connection [col.4, lines 60-65; col.12, lines 21-37; col. 16, lines 4-35; col.25, lines 32-34; i.e., the message corresponds to the outbound ACK packets and the inbound ACK

packets; both are processed within the CPD without involvement of the host CPU, wherein transmission of the inbound and outbound ACK occurs when the CPD has control of the TCP connection (because the CPU is not involved in these activities)].

Bennett does not specifically teach that said CPU providing to said CPD a media-access control (MAC) address.

However, it is well known in the art of Internet communication that in order for an Ethernet node (such as nodes of 106, Fig. 1) to communicate another node over the Internet, an ARP table must be established (at a gateway or in a local cache) to map each destined IP address to its corresponding MAC address. For each IP-MAC pair an ARP request is issued and a corresponding response containing an MAC address is collected. For example, in order for node 1000 (which resides in an ATM network) to communicate with node 276 (which resides in an Ethernet), an ARP table typically resides in node 272 to translate the IP address of node 276 to its MAC address. In order to build up the IP-MAC mapping for node 276, upon receiving a first packet received at node 272 that is destined for node 276, node 272 typically sends out an ARP request by broadcasting node 276's IP address to all the nodes residing in Ethernet 106. In response to the ARP request, only node 276 responds the request by sending its IP and MAC addresses to node 272 for establishing node 276's IP-MAC mapping in the ARP table.

Note that although in the specification Bennett only equips node 1000 (see Fig. 1) with the invented network interface card (NIC), it is obvious to an ordinary skilled artisan that any of the network nodes shown on Fig. 1 can be equipped with the same kind of NIC to receive the claimed speed advantage [See col. 20, line 65 - col. 21, line 2; claims 1-60, wherein the invented NIC and its associated methods are situated in a generic computer node residing on a generic network] . As such, it is clear that when the CPU of node 276 responds to an ARP request, its MAC and IP must be provided to the local NIC because the latter is the only contact point to the local Ethernet.

A. Bennett Does Not Disclose Several Limitations of Claim 1

Applicants respectfully disagree with the Final Rejection's construction of a "TCP connection" as "a physical connection established for transmission of a TCP packet." Such a construction conflates the physical layer (layer 1) of network communication protocols with the transport layer (layer 4), in which TCP resides. Moreover, a person having ordinary skill in the art (a "PHOSA") would know that networks do not function like the old-fashioned telephone system, where a fixed physical connection was established for each phone call. Furthermore, it would be impossible for "central processing unit (CPU) [10, Fig.3] running protocol processing instructions in the local host to create a TCP connection between the local host and the remote host," as alleged by the Final Rejection. CPU 10 has no way to establish a "physical connection" between the various networks, nodes and computers that lie between node 1000 and node 276 of FIG. 1 of Bennet, for example. Such a "physical connection" already exists, as shown in FIG. 1, and it was not "established" by "CPU 10."

Moreover, the Examiner's interpretation of a "TCP connection" as "a physical connection established for transmission of a TCP packet" contradicts Bennet and the understanding of a PHOSA. Bennet states: "Each computer shown in FIG. 1 includes the TCP/IP protocol stack, defined in the RFC documents," and refers in particular to the fundamental TCP specification, "RFC 793."[1] RFC 793 repeatedly refers to establishing a TCP connection, and leaves no doubt that the Examiner's construction of a "TCP connection" as "a physical connection established for transmission of a TCP packet" is unreasonable. For example, RFC 793 states:

> Connections:
>
> The reliability and flow control mechanisms described above require that TCPs initialize and maintain certain status information for each data stream. The combination of this information, including sockets, sequence numbers, and window sizes, is called a connection. Each connection is uniquely specified by a pair of sockets identifying its two sides.
>
> When two processes wish to communicate, their TCP's must first establish a connection (initialize the status information on each side).[2]

Moreover, RFC 793 also states, under the heading "Connection Establishment and Clearing," that: "A connection is fully specified by the pair of sockets at the ends."[3] It is well known, however, that TCP/IP packets may take different routes over various networks between their source and destination sockets. This flexibility in routing provided by packet-switched networks could not occur according to the Examiner's definition, which would require a fixed physical path for a "TCP connection." Thus, the Final Rejection's definition of a TCP connection contradicts Bennet as well as the TCP specification.

[1] Bennet, column 3, lines 45-51.

[2] RFC 793, page 5. For the Examiner's convenience, a copy of pages 5 and 10 of RFC 793 is enclosed, although the entire specification can be easily viewed at http://www.ietf.org/rfc/rfc0793.txt.

Another way to view the Final Rejection's definition of a "TCP connection" as a "physical connection" is that the Final Rejection ignores the limitation in claim 1 of "TCP," which by itself shows that the Final Rejection has failed to state a *prima facie* case of obviousness.

Nowhere in Bennet is the establishment of a TCP connection mentioned, in contrast to the limitation of claim 1 which recites "a central processing unit (CPU) running protocol processing instructions in the local host to create a TCP connection between the local host and the remote host." Moreover, as shown below under heading "II.B.," applying the teachings of Bennet would destroy a "TCP connection," (*assuming arguendo* that one would somehow exist) so it is unclear that Bennet teaches how to "create a TCP connection," as recited in claim 1.

Moreover, applicants respectfully assert that the Final Rejection's unreasonable construction of a "TCP connection" is contrived to attempt to find in Bennet other limitations that are also not present. For example, the Final Rejection of claim 1 quoted above asserts in part that "said CPD controls said connection." There is absolutely nothing in Bennet that teaches or suggests that the "network card 2000" of Bennet controls a TCP connection. The only TCP processing that Bennet says "TCP logic 93" of "network card 2000" does is checking for errors by checksumming and automatic generation of ACKs if the checksum indicate that data is corrupted. This is a far cry from the limitation of claim 1 that "said CPD controls said connection."

Curiously, even under the Final Rejection's unreasonable construction of a "TCP connection" as "a physical connection established for transmission of a TCP packet," the "network card 2000" would not have control of the various networks, nodes and computers that lie between node 1000 and node 276 of Bennet, any more than Bennet's "CPU 10" could have established that "physical connection." Thus, under both the Final Rejection's unreasonable construction of a "TCP connection" as "a physical connection established for transmission of a TCP packet," and under the definition of a TCP connection provided for example in RFC 793, there is no indication anywhere in Bennet that the "network card 2000" of Bennet controls a "TCP connection."

[3] RFC 793, page 10.

The Final Rejection proceeds further out on a limb by stating: "Since TCP process 91 of Fig. 2B is designed for constructing a TCP packet, the execution of process 91 is in a way "running protocol processing instructions.'" That is, the Final Rejection alleges that "constructing a TCP packet" by Bennett equates to "running protocol instructions in the local host to create a TCP connection between the local host and the remote host," as recited by claim 1. If that were true, construction of the packet would somehow cause distal network routers to physically contact each other to form a new "physical connection." Moreover, every time a new packet was constructed a new TCP connection would be created, even for successive packets constructed from contiguous application data. Such an interpretation is untenable. Furthermore, such an interpretation contradicts the Examiner's later allegation that Bennett's automatic ACK generation equates to controlling the TCP connection by Bennett's "network card 2000", because there is no indication that the ACK corresponds to the TCP connection allegedly created by constructing a packet in the "TCP process 91" of "local node 1000". Indeed, by the Examiner's own reasoning, the construction of an ACK packet would create a new TCP connection,[4] so the ACK could not correspond to the TCP connection that the Examiner alleges was created by constructing a packet in the "TCP process 91," in contrast to another limitation of claim 1. Appellants respectfully submit that for these reasons, as well as others discussed below, the Final Rejection's allegation of what Bennet teaches regarding claim 1 is at best unreasonable.

Applicants respectfully submit that because Bennet differs so markedly and in so many ways from claim 1, all of which were presented to the Examiner in the Appeal in which he was Reversed, and because the Final Rejection offers no clue of how or why a PHOSA would change Bennet to meet the limitations of claim 1, the Examiner fails, yet again, to present a *prima facie* case of anticipation or obviousness.

[4] The Examiner's Answer on page 10, lines 1-4 states that an ACK packet "by itself is a TCP packet", but fails to realize that by the Examiner's logic, constructing that ACK packet by "network card 2000" would create a new TCP connection that is different than the TCP connection the Examiner alleges was created earlier by constructing a TCP packet in "TCP process 91" of "local node 1000".

B. <u>A PHOSA Would Not Have Modified Bennett as Proposed by the Final Rejection to Attempt to Meet Another Missing Limitation of Claim 1</u>

The Final Rejection grudgingly admits, in response to the complete Reversal of the previous Final Rejection, that "Bennett does not specifically teach that said CPU providing to said CPD a media-access control (MAC) address." However, the latest Final Rejection alleges "it is obvious to an ordinary skilled artisan that any of the network nodes shown on Fig. 1 can be equipped with the same kind of NIC to receive the claimed speed advantage." Applicants respectfully disagree.

A PHOSA would not have modified Bennet as suggested by the Final Rejection because such a PHOSA would understand that Bennet's automatic ACK generation would destroy the reliability and guaranteed delivery of data essential to TCP. For this reason, a PHOSA would have avoided equipping any NIC having Bennet's claimed speed advantage, in contrast to the Final Rejection's proposed modification of Bennet.

Bennett claims to automatically send an acknowledgement (an "ACK") for a datagram upon verifying the checksum for the datagram.[5] But the TCP protocol specifies that sending an ACK signals to the receiver of the ACK that all the data prior to that ACK number has been successfully received by the sender of the ACK. Thus, the automatic generation of an ACK by Bennett may cause errors, because it signals that all previous data has been successfully received, when in fact it may not have been successfully received. Such errors would destroy the reliability and guaranteed delivery of data provided by TCP.

As noted in Stevens, "TCP/IP Illustrated, Volume 1, The Protocols" (hereinafter "Stevens"), "the acknowledgement number in the TCP header means that the sender has successfully received up through but not including that byte. There is currently no way to acknowledge selected pieces of the data stream. For example, if bytes 1-1024 are received OK, and the next segment contains bytes 2049-3072, the receiver cannot

[5] See, e.g., Bennett, column 12, lines 7-11; column 16, lines 19-26. This aspect of Bennett is described in the Final Rejection of 01/18/06 on page 3, lines 15-17, which states: "when a system uses network card 2000 (of Fig.3) to replace the related processing that is otherwise handled by TCP/IP software, the message related to TCP ACK is totally processed at the CPD."

acknowledge this new segment. All it can send is an ACK with 1025 as the acknowledgement number."[6]

According to Bennett's preferred embodiment, however, the "NIC 2000" would automatically send an ACK with 3073 as the acknowledgement number for the example of Stevens, assuming that the checksum for bytes 2049-3072 was valid. This would indicate to the receiver of that ACK that all data up through byte 3072 was successfully received by the sender of the ACK, even though bytes 1025-2048 were never in fact received.

Because Bennett makes no provision for resending lost packets, and the sender of data would believe that no prior packets need to be resent once the sender has received the ACK that would be automatically generated according to Bennett's disclosure, Bennett's preferred embodiment would cause the loss and corruption of data. In other words, Bennett's automatic sending of an ACK upon verifying the checksum for a datagram violates both the rules and the purpose of the TCP protocol.

In the event that a packet is lost, TCP provides a retransmission mechanism, but Bennett would defeat this mechanism as well. TCP retransmission depends upon the failure of the sender of data to receive an ACK within a certain time period, and Bennett thwarts this retransmission mechanism by automatically sending ACKs despite not having received all the data. As stated by Stevens:

> TCP provides a reliable transport layer. One of the ways it provides reliability is for each end to acknowledge the data it receives from the other end. *But data segments and acknowledgements can get lost. TCP handles this by setting a timeout when it sends data, and if the data isn't acknowledged when the timeout expires, it retransmits the data.*[7]

Stevens also notes that TCP uses a sliding window protocol to send multiple packets before waiting for an acknowledgement, as is well known to a PHOSA. The

[6] Richard Stevens, "TCP/IP Illustrated, Volume 1, The Protocols" (1994), page 226, lines 34-38. A copy of page 226 of Stevens was enclosed with the Appeal Brief and the Amended Appeal Brief, which quoted and discussed Stevens. Page 226 of Stevens was also enclosed, quoted and discussed in the Request for Reconsideration filed February 9, 2006, which was noted to have been considered by the Examiner in an Advisory Action dated February 28, 2006.

[7] Stevens, page 297, lines 2-6, emphasis added. A copy of page 297 of Stevens was enclosed with the Appeal Brief and the Amended Appeal Brief, which quoted and discussed that page. Page 297 of Stevens was also enclosed, quoted and discussed in the Second Request for Reconsideration filed March 15, 2006, which was noted to have been considered by the Examiner in an Advisory Action dated April 3, 2006.

sliding window protocol, which has been part of TCP since its inception decades ago, provides a mechanism to communicate multiple packets between acknowledgments without overloading the receiver. As stated by Stevens:

> In Chapter 15 we saw that TFTP uses a stop-and-wait protocol. The sender of a data block required an acknowledgement for that block before the next block was sent. In this chapter we'll see that TCP uses a different form of flow control called a *sliding window* protocol. It allows the sender to transmit multiple packets before it stops and waits for an acknowledgement. This leads to faster data transfer, because the sender does not have to stop and wait for an acknowledgement each time a packet is sent.[8]

In other words, unlike the TFTP protocol, TCP does not stop and wait after each packet has been sent to receive an acknowledgment before sending the next packet. Note also that Bennett's alleged automatic ACK generation would not be known to the sender of data, which would send multiple packets according to TCP's sliding window protocol, not knowing that Bennett's system can only receive one packet at a time without errors. There is no teaching in Bennett of requiring the remote sender to send a single packet and then stop and wait for an acknowledgment from Bennett's device before sending the next packet. *Assuming arguendo* that the data sender would implement such a stop-and-wait policy, TCP communications would be dramatically slowed, analogous to requiring jet airplanes to carry only a single passenger at a time, with all the other passengers waiting in line for the jet to return from its round trip. Of course, Bennett does not teach or suggest modifying TCP to employ such a stop-and-wait procedure. Instead, Bennett states that "It is an object of the present invention to provide an improved method and apparatus for efficiently operating a reliable communication protocol in a computer network,"[9] and claims that its "Internet Accelerator"..."results in a significant improvement in system performance over systems according to the prior art."[10]

In accordance with the above explanation from Stevens is Douglas E. Comer, "Internetworking with TCP/IP" (hereinafter "Comer"), which begins by discussing

[8] Stevens, page 275, lines 3-8, emphasis in original. Page 275 of Stevens was first quoted and discussed in the Interview Summary and Request for Reconsideration filed April 11, 2006, which was noted to have been considered by the Examiner in an Advisory Action dated April 24, 2006. Page 275 of Stevens was also enclosed, quoted and discussed in the Appeal Brief and the Amended Appeal Brief.

[9] Bennett, column 1, line 66 – column 2, line 1.

[10] Bennett, column 8, line 1 and column 2, lines 17-18.

elements of retransmission for "*most reliable protocols*,"[11] allowing the student to first understand the basics of retransmission before learning the more complicated function of sliding windows.

Comer begins discussing the sliding window protocol of TCP later, on page 175, by stating:

> **12.5 The Idea Behind Sliding Windows**
>
> Before examining the TCP stream service, we need to explore an additional concept that underlies stream transmission. The concept, known as a *sliding window*, makes stream transmission efficient...
>
> *A simple positive acknowledgement protocol wastes a substantial amount of network bandwidth because it must delay sending a new packet until it receives an acknowledgement for the previous packet.*[12]

The fact that TCP uses sliding windows and sends out multiple packets before receiving an acknowledgement for the first packet is well known to those of ordinary skill in the art and discussed, for example, on page 187 of Comer, which states:

> *Acknowledgements always specify the sequence number of the next octet that the receiver expects to receive.*
>
> The TCP acknowledgement scheme is called *cumulative* because it reports how much of the stream has accumulated. Cumulative acknowledgements have both advantages and disadvantages. One advantage is that acknowledgements are both easy to generate and unambiguous. Another advantage is that lost acknowledgements do not necessarily force retransmission. A major disadvantage is that <u>the sender does not receive information about all successful transmissions, but only about a single position in the stream that has been received.</u>[13]

Later on the same page, Comer states:

> Because (TCP) segments travel in IP datagrams, they can be lost or delivered out of order; the receiver uses the sequence number to reorder segments.[14]

This statement also makes clear that TCP does not wait to receive an acknowledgement for each packet before sending the next packet, because the packets

[11] Douglas E. Comer, "Internetworking with TCP/IP," Volume 1, (1991), page 173, line 38, emphasis added. A copy of pages 173-175 and 187 of Comer was enclosed with the Appeal Brief and with the Amended Appeal Brief. Pages 173-175 and 187 of Comer were first enclosed, quoted and discussed in the Second Request for Reconsideration filed March 15, 2006, which was noted to have been considered by the Examiner in an Advisory Action dated April 3, 2006.
[12] Comer, page 175, lines 19-32, emphasis in original.
[13] Comer, page 187, lines 16-40, italics in original, underline added.

could in that hypothetical example never arrive out of order. Even Bennett recognizes that TCP segments can be received out of order,[15] although it fails to recognize that its automatic ACK generation would destroy the basic functions and reliability of TCP.

As noted previously, Bennett claims to automatically send an ACK for a datagram upon verifying the checksum for the datagram.[16] But the TCP protocol specifies that sending an ACK signals to the receiver of the ACK that all the data prior to that ACK number has been successfully received by the sender of the ACK. Because Bennett sends ACKs despite not having received all the data signified by its ACKs, the retransmission timeout does not expire, and the lost data is not retransmitted. Stated differently, sending an ACK upon receiving a packet even though a prior packet may not have been received violates the TCP protocol. Moreover, because Bennett teaches automatically generating ACKs upon verification of a checksum, the ACKs signaling to the sender that all prior data in the stream has been received without regard to whether this is true, Bennett's ACKs would circumvent the timeout and retransmission mechanism that TCP relies upon, causing data corruption.

In addition to the glaring failures of Bennett noted above, a PHOSA would have recognized other failures of Bennett's ACK generation mechanism. For example, because Bennett's card 2000 automatically generates ACKs, it neglects other aspects of TCP's requirement of assured delivery of data. This is because the card 2000 at most performs partial protocol processing despite sending ACKs, and even for checksummed packets that arrived in order there is still the opportunity for Bennett's CPU 10 to discard a packet due to other reasons, in which case the ACK sent by the card 2000 is erroneous. For instance, header errors that escape detection by the simple checksum mechanism would presumably be recognized by CPU 10, causing the associated data to be discarded. Conversely, should CPU 10 or TCP Process 91 crash after an ACK had been automatically generated by card 2000 but before that data could be processed by TCP Process 91, that data would be lost and not retransmitted. Similarly, the TCP Process 91 running on CPU 10 may also discard the data due to resource limitations, with TCP

[14] Comer, page 187, lines 6-8.
[15] Bennett, column 11, line 66 – column 12, line 2.
[16] See, e.g., column 12, lines 7-11; column 16, lines 19-26 of Bennett.

Process 91 assuming that no ACK for the data has been sent and that the data would be retransmitted once a timeout occurs at the sender.

A primary purpose of the TCP protocol is the guaranteed delivery of data. Bennett's foremost objective is also to "provide an improved method and apparatus for efficiently operating a reliable communication protocol in a computer network."[17] Yet the invention actually taught by Bennett would destroy the reliability and guaranteed delivery of data, thwarting the primary purposes of TCP and Bennett. For at least these reasons, a PHOSA would not have used Bennet, and further would not have modified Bennet as proposed by the Final Rejection to add additional dysfunctional hardware to further destroy TCP.

Indeed, Bennett instead demonstrates a long-standing need for the invention defined by the present claims and a failure of others in their approach to solving that need, which is the foremost objective indication of nonobviousness.[18]

For all the above reasons, the Final Rejection has not presented a *prima facie* case of obviousness of claim 1.

C. <u>*Assuming Arguendo* that a PHOSA would have Modified Bennett as Proposed by the Final Rejection, the Result would have been Substantially Different than Claim 1</u>

Section "II.A." above lists a number of ways in which Bennet is substantially different than claim 1. The Final Rejection provides no indication of how or why a PHOSA would have modified Bennet to try to be closer to claim 1, despite those differences being discussed repeatedly in prior applicant responses. For at least those reasons, the Final Rejection fails to prevent a *prima facie* case of obviousness.

Section "II.B." above shows that a PHOSA would not have modified Bennet as proposed by the Final Rejection, because Bennet destroys the primary attributes of TCP. *Assuming arguendo* that a PHOSA would have modified Bennett as proposed by the Final Rejection, the result would have been substantially different than claim 1 for

[17] Bennett's Summary of the Invention, column 1, line 65 – column 2, line 1.
[18] The previous Final Rejection included an obviousness rejection over Bennet in view of U.S. Patent No. 6,173,333 to Jolitz et al., which was withdrawn after applicants demonstrated that Jolitz too was

reasons additional to those mentioned in section "II.A." In particular, because Bennet destroys the reliability and guaranteed delivery of data that are the hallmarks of TCP as discussed in section "II.B.," Bennet would not have provided the synergistic combination of reliability and speed that is afforded by the invention defined in claim 1. Instead, in addition to all the various significant ways in which claim 1 differs from Bennet as discussed in "II.A.," modifying Bennet as proposed by the latest Final Rejection would result in various errors that violate the fundamental purpose TCP, such that the proposed modification would differ in another extremely important way from claim 1.

That is, other network mechanisms and layers are known to have errors, because TCP guarantees that despite those errors, data delivered according to TCP will be error free. The modification proposed by the Final Rejection would break that guarantee, in contrast to the invention defined by claim 1. For this additional and considerable reason, claim 1 is nonobvious over the Final Rejection's proposed modification of Bennet.

III. Conclusion

In the interest of brevity, applicants will not in this Request provide arguments that respond to the rejections of claims other than claim 1. The interested reader is referred to the Amended Appeal Brief and Reply Brief, which contain numerous additional points that may have to be repeated yet again in another brief or in District Court. Applicants respectfully assert the latest Final Rejection has once again failed to present a *prima facie* case of anticipation or obviousness. Applicants urge the Examiner to reconsider this Rejection, and assert that the claims are in condition for allowance.

Respectfully submitted,

[illegible]

Mark Lauer
Reg. No. 36,578
6601 Koll Center Parkway
Suite 245
Pleasanton, CA 94566
Tel: (925) 621-2121
Fax: (925) 621-2125

CERTIFICATE OF ELECTRONIC TRANSMISSION
I hereby certify that this correspondence is being transmitted via the US Patent Office electronic filing system in accordance with § 1.6(a)(4)and addressed to: MS Petition, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on October 6, 2008.

Date: 10-6-08 [illegible]
Mark Lauer

nonenabled, but which provides another example of the failure of others to accelerate TCP while maintaining its primary attributes of reliability and guaranteed delivery of data.

Multiplexing:

To allow for many processes within a single Host to use TCP communication facilities simultaneously, the TCP provides a set of addresses or ports within each host. Concatenated with the network and host addresses from the internet communication layer, this forms a socket. A pair of sockets uniquely identifies each connection. That is, a socket may be simultaneously used in multiple connections.

The binding of ports to processes is handled independently by each Host. However, it proves useful to attach frequently used processes (e.g., a "logger" or timesharing service) to fixed sockets which are made known to the public. These services can then be accessed through the known addresses. Establishing and learning the port addresses of other processes may involve more dynamic mechanisms.

Connections:

The reliability and flow control mechanisms described above require that TCPs initialize and maintain certain status information for each data stream. The combination of this information, including sockets, sequence numbers, and window sizes, is called a connection. Each connection is uniquely specified by a pair of sockets identifying its two sides.

When two processes wish to communicate, their TCP's must first establish a connection (initialize the status information on each side). When their communication is complete, the connection is terminated or closed to free the resources for other uses.

Since connections must be established between unreliable hosts and over the unreliable internet communication system, a handshake mechanism with clock-based sequence numbers is used to avoid erroneous initialization of connections.

Precedence and Security:

The users of TCP may indicate the security and precedence of their communication. Provision is made for default values to be used when these features are not needed.

Transmission is made reliable via the use of sequence numbers and acknowledgments. Conceptually, each octet of data is assigned a sequence number. The sequence number of the first octet of data in a segment is transmitted with that segment and is called the segment sequence number. Segments also carry an acknowledgment number which is the sequence number of the next expected data octet of transmissions in the reverse direction. When the TCP transmits a segment containing data, it puts a copy on a retransmission queue and starts a timer; when the acknowledgment for that data is received, the segment is deleted from the queue. If the acknowledgment is not received before the timer runs out, the segment is retransmitted.

An acknowledgment by TCP does not guarantee that the data has been delivered to the end user, but only that the receiving TCP has taken the responsibility to do so.

To govern the flow of data between TCPs, a flow control mechanism is employed. The receiving TCP reports a "window" to the sending TCP. This window specifies the number of octets, starting with the acknowledgment number, that the receiving TCP is currently prepared to receive.

2.7. Connection Establishment and Clearing

To identify the separate data streams that a TCP may handle, the TCP provides a port identifier. Since port identifiers are selected independently by each TCP they might not be unique. To provide for unique addresses within each TCP, we concatenate an internet address identifying the TCP with a port identifier to create a socket which will be unique throughout all networks connected together.

A connection is fully specified by the pair of sockets at the ends. A local socket may participate in many connections to different foreign sockets. A connection can be used to carry data in both directions, that is, it is "full duplex".

TCPs are free to associate ports with processes however they choose. However, several basic concepts are necessary in any implementation. There must be well-known sockets which the TCP associates only with the "appropriate" processes by some means. We envision that processes may "own" ports, and that processes can initiate connections only on the ports they own. (Means for implementing ownership is a local issue, but we envision a Request Port user command, or a method of uniquely allocating a group of ports to a given process, e.g., by associating the high order bits of a port name with a given process.)

A connection is specified in the OPEN call by the local port and foreign socket arguments. In return, the TCP supplies a (short) local